.NET Framework Security

Brian LaMacchia
Development Lead
.NET Framework Security
Microsoft Corporation

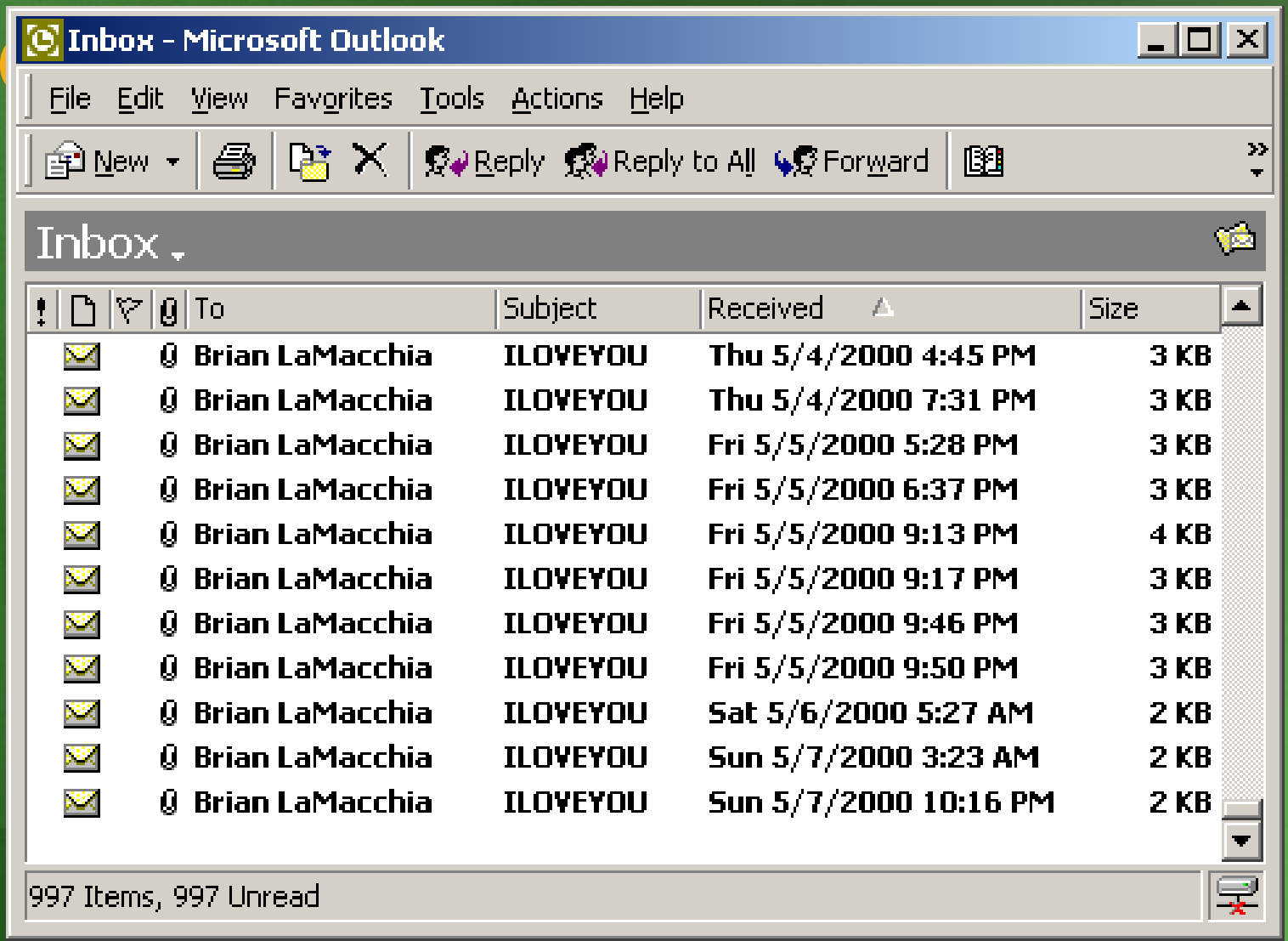
3-334


Microsoft®
PDC 2000
Professional Developers Conference
the defining point

Agenda

- Motivation & Design Goals
- Permissions & Walking the Stack
- The “Evidence-Based Security” model
- Quick Tour of Security Classes
- Signed XML

Why Should You Care About Platform S



Design Goals

- Provide a robust security system for partially-trusted, mobile code
- Make it easy to:
  - Express fine-grained authorizations
  - Extend & customize the system
  - Perform security checks in user code
- No end-user UI!
  - Never ask a user to make a

Agenda

- Motivation & Design Goals
- Permissions & Walking the Stack
- The “Evidence-Based Security” model
- Quick Tour of Security Classes
- Signed XML

Definitions

- **Authentication**
  - **Determining the identity of the party/entity making a request.**
  - **ASP+ Security, Session 4-433**
- **Authorization**
  - **Determining whether to honor a request made by an identified party.**
  - **Ex: Sending mail on behalf of a user**
  - **ASP+ Security & this session**

Permissions

- A **permission** object represents a specific authorization
  - Example: Read access to c:\temp directory
- A permission **grant** is an authorization given to an assembly
- A permission **demand** is a security check for corresponding grants
- Frameworks include permissions for common resources
  - Examples: FileIOPermission, SocketPermission, UIPermission, PrincipalPermission

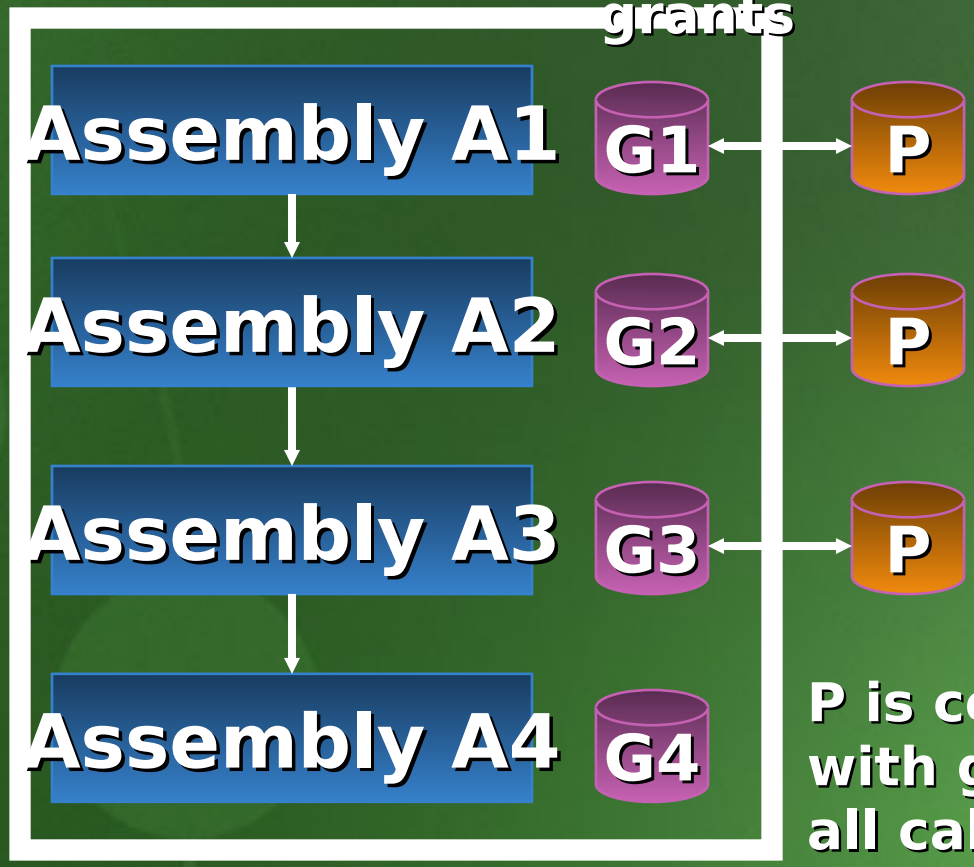
Code Access Security

- Most permissions are code access permissions
  - Demanding a permission performs a stack walk checking for related grants
  - Support dynamic stack modifiers
  - Two ways to make checks:
    - Imperatively (method implementation)
    - Declaratively (method metadata)

Stack Walk Behavior

Each assembly has a set of corresponding grants

Call Stack Grows Down



Method in Assembly A4 demands permission P

P is compared with grants of all callers on the stack above A4

Stack Walk Modifiers

- Modifiers provide fine-grained, dynamic control over state of grants on the stack
- perm.Assert()
  - “I vouch for my callers; checks for perm can stop at this frame”
- Example: “Gatekeeper” classes
  - Managed wrappers for unmanaged resources
    - Demand appropriate permission from caller
    - Assert permission to call unmanaged code

Imperative Security Checks

- Example: Frameworks File object constructor
  - Requires “read” access to the corresponding file

```
public File(String fileName) {
  // Must fully qualify the path for the security check
  String fullPath =
    Directory.GetFullPathInternal(fileName);
  new
    FileIOPermission(FileIOPermissionAccess.Read,
    fullPath).Demand();
  //[ ... rest of function ... ]
```

Declarative Security Checks

- Declarative demands
  - Part of a method's metadata
  - Implemented with custom attributes
  - JIT converts the annotation to a check
- Example (C#):

```
[FileIOPermission(SecurityAction.Demand, Write =
      "c:\\temp")]
public void foo() {
  // class does something with c:\temp
}
```

Agenda

- Motivation & Design Goals
- Permissions & Walking the Stack
- The “Evidence-Based Security” model
- Quick Tour of Security Classes
- Signed XML

Evidence-Based Security

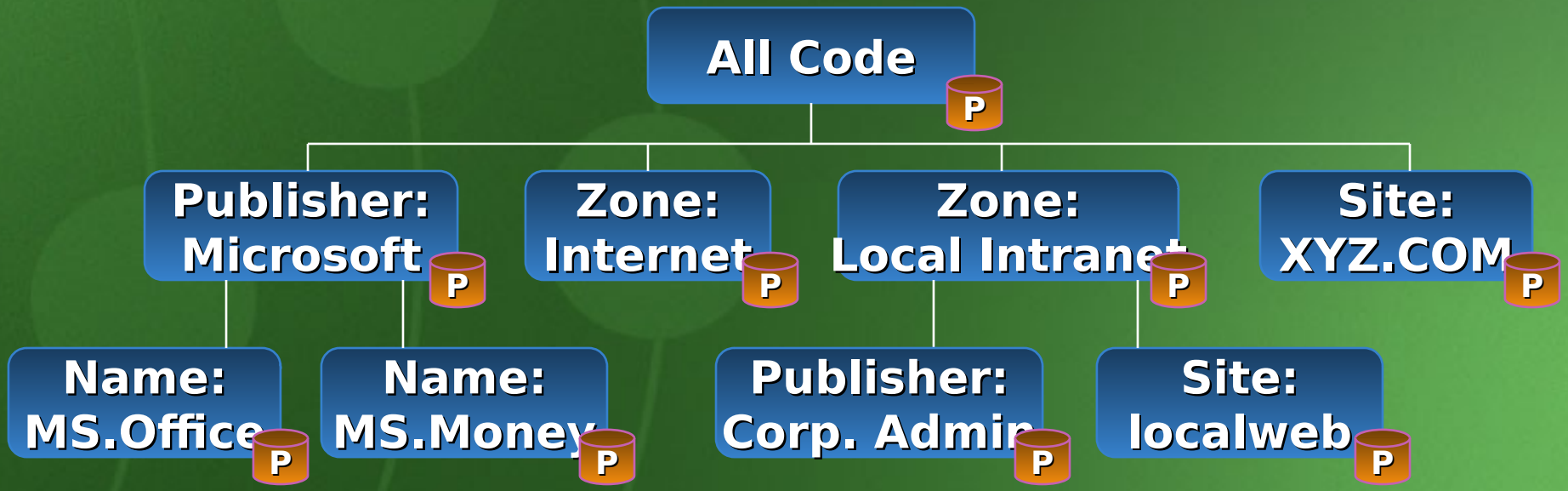
- Permissions
  - Objects that represent specific authorizations
- Policy
  - Determines set of permissions to grant to an assembly
- Evidence
  - Inputs to policy, from multiple sources
- All three are fully extensible

Policy

- **Policy** is the process of determining the permissions to grant to code
  - Permissions granted to code, not user
  - Grants are on a per-assembly basis
- Multiple levels of policy
  - Machine-wide, User-specific by default
  - Further policy restrictions allowed on a per-application domain basis

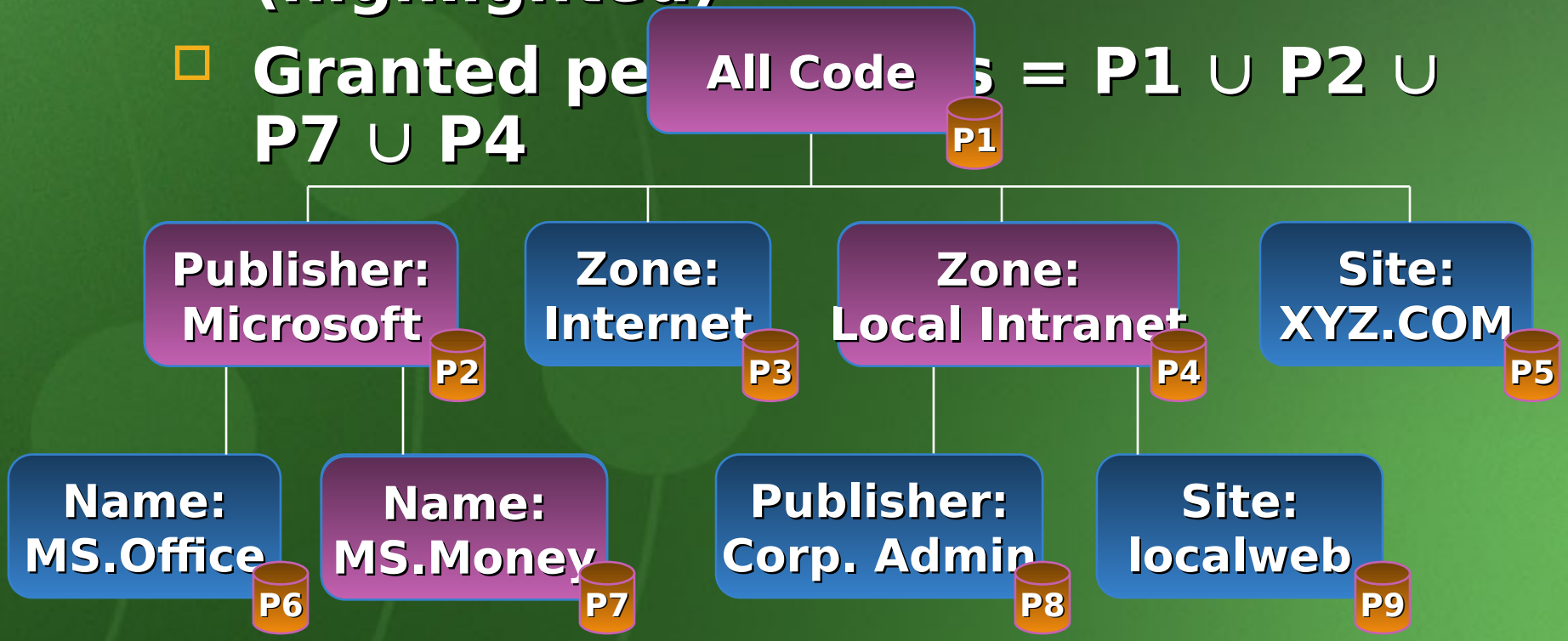
Inside policy

- Each policy level is a collection of code groups
  - Code has identity in the runtime, just like users have identity in Windows NT®
  - Permissions are associated with each code group
- Evidence determines group membership
  - In the group, get granted the related permissions



Sample Policy Level

- Example: MS.Money on Local Intranet
  - Member of four groups (highlighted)
  - Granted pe s = P1 ∪ P2 ∪ P7 ∪ P4



Evidence

- **Evidence** is the input to policy
- Example: Info about a code assembly
  - Shared names } cryptographically strong
  - Publisher identity
  - Location of origin (URL, zone, site)
- Evidence is completely extensible
  - Any object can be a piece of evidence
  - Only impacts grants if there is a

Host & App Input To Policy

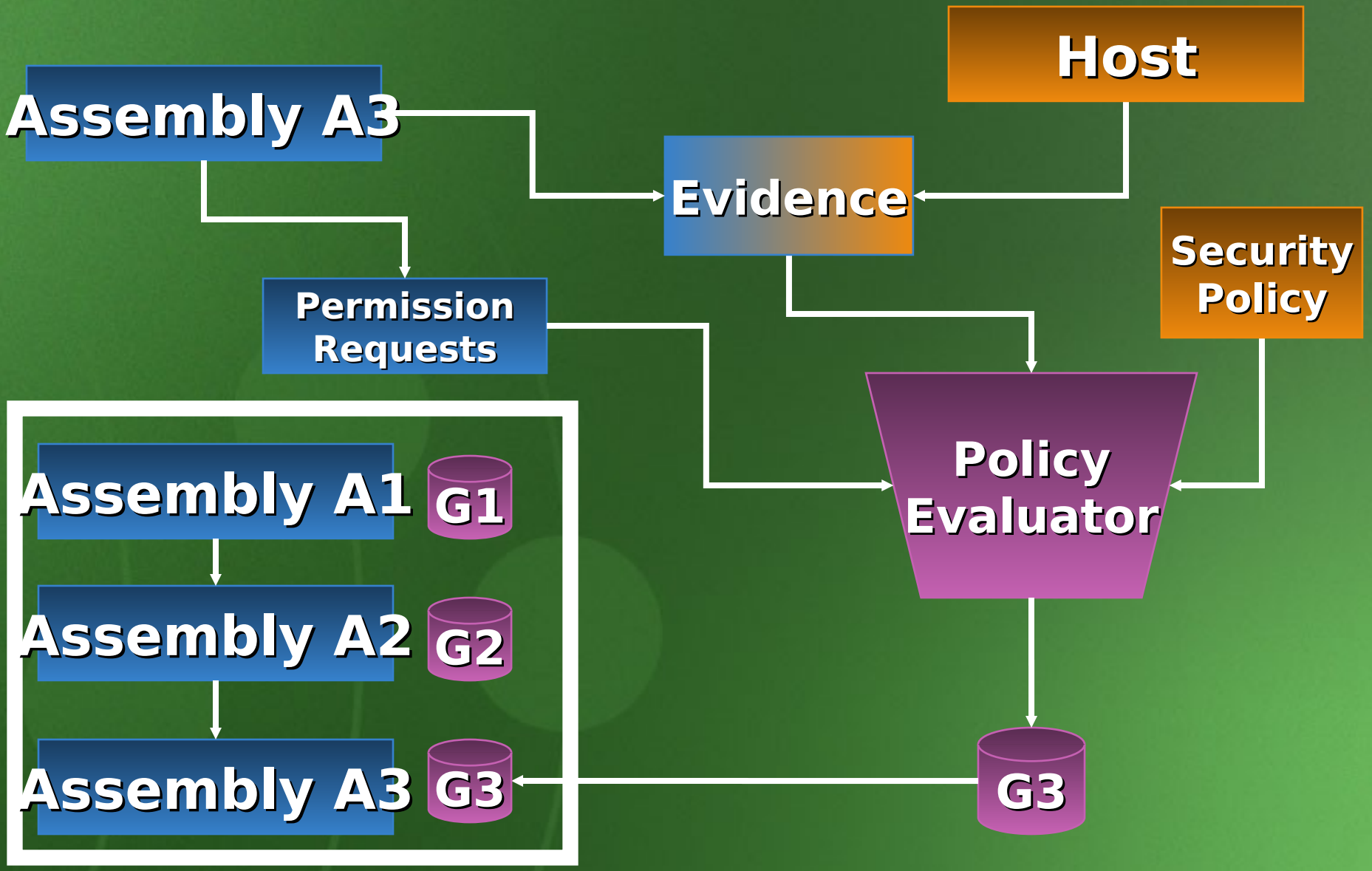
- Hosts can influence policy
  - Hosts specify “implicitly trusted” evidence
  - Custom membership conditions can interface with other authorization systems
  - Ex: ASP+/ISP application hosting
- Application domains can limit policy for other domains they create
  - Ex: SQL Server™ user-defined assemblies

Assembly Input To Policy

- Assemblies can **request** permissions
  - Three types of request: Minimal, Optional, Refuse
  - If policy does not grant everything in the "Minimal" set, assembly fails to load
  - Assembly is granted:

    (MaxAllowed ∩ (Minimal ∪ Optional)) – Refused
- Assemblies can carry evidence, too
  - Assembly evidence is "initially untrusted"
  - Policy evaluates assembly evidence and decides whether to use it
  - Example: third-party certifications

Putting It All Together



Agenda

- Motivation & Design Goals
- Permissions & Walking the Stack
- The “Evidence-Based Security” model
- Quick Tour of Security Classes
- Signed XML

Permissions

System.Security.Permissions.<class>

- Individual Permission classes
  - E.g., FileIOPermission
- *PermissionAttribute
  - Declarative attributes, one per permission (FileIOPermissionAttribute)
  - One mandatory argument (action type)
    - SecurityAction.Demand

Isolated Storage

System.IO.IsolatedStorage.<class>

- IsolatedStorageFile class
  - Filesystem abstraction for local store
- IsolatedStorageFileStream
  - Data stream in the filesystem
- Two sample applications on the CD
  - IsoStorage1: Assembly-specific storage
  - IsoStorage2: Application-specific

Infrastructure, Policy And Principal

- System.Security.<class>
  - Core infrastructure classes
  - PermissionSet
- System.Security.Policy.<class>
  - ICodeGroup
  - IMembershipCondition
- System.Security.Principal.<class>
  - Role-based security objects

Cryptography Classes

System.Security.Cryptography.<class>

- **Core cryptography classes**
- **SymmetricAlgorithm**
  - **TripleDES, DES, RC2 (abstract)**
  - **TripleDES_CSP, etc. (implementation)**
- **AsymmetricAlgorithm**
  - **RSA, DSA (abstract)**
  - **RSA_CSP, DSA_CSP (implementation)**
- **Demo in "Handling the Basics..."**

Agenda

- Motivation & Design Goals
- Permissions & Walking the Stack
- The “Evidence-Based Security” model
- Quick Tour of Security Classes
- Signed XML

Signed XML Classes

Signed XML classes

- System.Security.Cryptography.XML.<class>
  - Support for IETF/W3C XMLDSIG proposed standard
  - Simplified operations
    - SignedXML class
  - Base XMLDSIG object classes
    - Signature, SignedInfo
    - Reference
    - KeyInfo, KeyValue

Signed XML Sample

- XMLDSIG supports three methods of signing an XML element
  - Wrapped, embedded, and detached
  - Wrapped works like CMS (S/MIMEv3), the signature is wrapped around the element
- XMLDSIG is key-centric
  - No certificates required
- Element to sign:

How To Sign Some XML

```
public IElement Sign(AsymmetricAlgorithm key, IElement element) {
// Set up the SignedXML object and associate w/ signing key
SignedXml signedXml = new SignedXml();
signedXml.SigningKey = key;
// Create a data object to hold the data we wish to sign
DataObject dataObject = new DataObject();
dataObject.Embedded = true;
dataObject.Data = element;
dataObject.Id = "ID1";
  // Add a hash reference to the data object
  Reference reference = new Reference();
  reference.DataObject = dataObject;
  signedXml.AddReference(reference);
  // Compute hashes & the digital signature
  signedXml.ComputeSignature();
     return(signedXml.Xml);   // send back the resulting
   } XML
```

The Signed Sample

```
<Signature>
 <SignedInfo>
<SignatureMethod Algorithm="http://www.w3.org/2000/02/xmldsig#rsa-sha1">
</SignatureMethod>
      <Reference URI="#ID1">
        <Transforms>
              <Transform Algorithm="http://www.w3.org/TR/1999/WD-xml-
        c14n-19991115">
              </Transform>
<DigestMethod>http://www.w3.org/2000/02/xmldsig#sha1</DigestMethod>
  <DigestValue>KwPji6BszJTYHoRXHhSd1f58Wl0=</DigestValue>
      </Reference>
 </SignedInfo>
   <SignatureValue>

   /fijURqKNWNH1x9En5KlhgaWh5Jbu5Ht0FGiADJB5fsrYt7AY/S7ossEhkHjV
   9D
       szinRSUMc3g6js+pUI6fJIw==
   </SignatureValue>
   <Object Id="ID1">
   <MyElement>MyData</MyEleme
   nt>
</Signature>
   </Object>
```

Summary

- Code Access Security
  - Authorization (permission) enforcement mechanism for partially-trusted code
- Evidence-based Security
  - Policy model for granting permissions
  - Lots of new functionality and flexibility for clients & servers
  - Extensibility & customization available throughout the model
- Signed XML
  - W3C/IETF proposed standard
  - Supported in the Frameworks

Related Sessions And References

- Related sessions:
  - 4-433: Active Server Pages+ Security
  - 3-222: Handling the Basics: Getting the most out of the Frameworks
  - 1-222: Security Considerations for Download Controls
  - 2-336: SQL Server™ and .NET Platform
- References:

Questions?

Where do you want to go today?

Microsoft